Testing Java code using QuickCheck and JavaErlang

Lars-Åke Fredlund

Facultad de Informática
Universidad Politécnica de Madrid

ProTest
property based testing

Tutorial Structure

- A new library **JavaErlang** for calling Java code from Erlang
- Using **JavaErlang** and **QuickCheck** to test Java code

Calling Java from Erlang: what exists?

- JInterface – In Erlang/OTP
- Erjang – Erlang on the JVM
- erlang4j – not alive?

JInterface

- In Erlang/OTP
- Provides Java nodes which can communicate with Erlang nodes

JInterface

- In Erlang/OTP
- Provides Java nodes which can communicate with Erlang nodes
- Idea: Erlang sends Erlang terms, and Java sends Erlang terms coded as Java objects

JInterface

- In Erlang/OTP
- Provides Java nodes which can communicate with Erlang nodes
- Idea: Erlang sends Erlang terms, and Java sends Erlang terms coded as Java objects
- Provides operations for letting Java create Erlang values, and taking apart Erlang values:

```
OtpErlangAtom a = new OtpErlangAtom("hello");
String s = a.atomValue();
```

JInterface

- In Erlang/OTP
- Provides Java nodes which can communicate with Erlang nodes
- Idea: Erlang sends Erlang terms, and Java sends Erlang terms coded as Java objects
- Provides operations for letting Java create Erlang values, and taking apart Erlang values:

```
OtpErlangAtom a = new OtpErlangAtom("hello");
String s = a.atomValue();
```

- Java and Erlang have to agree on a shared vocabulary typically communicated as tuples:
  `mk_tree`, `{add_root,Element,Tree}`, `{parent,Tree}`, ...

JInterface in practice

- A lot of Java code to unpack/pack data to/from Erlang:

```
do {
  OtpErlangObject msg = msgs.receive();
  if (msg instanceof OtpErlangTuple) {
    OtpErlangTuple tuple = (OtpErlangTuple) msg;
    if (tuple.arity() == 3 &&
        tuple.elementAt(0).instanceof OtpErlangAtom) {
      String tag =
        ((OtpErlangAtom) tuple.elementAt(0)).atomValue(
      OtpErlangPid replyPid =
        (OtpErlangPid) tuple.elementAt(1);

      if (tag.equals("mk_tree"))) { /* ... */ ; }
      else if (tag.equals("add_root")) { /* ... */ ; }
      } else /* ... */ ;
  ...
}
```

- How to communicate Java references?

Design goals for JavaErlang

Design goals for JavaErlang

- Use JInterface

Design goals for JavaErlang

- Use JInterface
- All public Java methods callable without prior agreement between Java and Erlang; also access to public attributes

Design goals for JavaErlang

- Use JInterface
- All public Java methods callable without prior agreement between Java and Erlang; also access to public attributes
- No writing of Java code necessary at all

Design goals for JavaErlang

- Use JInterface
- All public Java methods callable without prior agreement between Java and Erlang; also access to public attributes
- No writing of Java code necessary at all
- Permit communication of Java references

In practice:

In practice:

- Starting Java:

```
{ok,Node}= java:start_node().
```

In practice:

- Starting Java:
  `{ok,Node}= java:start_node().`
- A Java class `p1.p2.c` gets translated to an Erlang module `p1_p2_c`.

In practice:

- Starting Java:
  `{ok,Node}= java:start_node().`
- A Java class `p1.p2.c` gets translated to an Erlang module `p1_p2_c`.
- Two styles of invoking Java: (i) using the generated Erlang module or (ii) using the Erlang `java` module API

In practice:

- Starting Java:
  `{ok,Node}= java:start_node().`
- A Java class `p1.p2.c` gets translated to an Erlang module `p1_p2_c`.
- Two styles of invoking Java: (i) using the generated Erlang module or (ii) using the Erlang `java` module API
- Example: creating a new instance using `new p1.p2.c(Args)` becomes
  - using the Erlang module:

    ```
    p1_p2_c:c(NodeId,Args)
    ```

  - using the Erlang `java` module API:

    ```
    java:new(NodeId,'p1.p2.c',[Args])
    ```

In practice: method calls and attribute access

- A call `Obj.meth(Args)` to an instance method of an object, when `Obj` is of class `p1.p2.c` becomes
  - `p1_p2_c:meth(Obj,Args)` or
  - `java:call(Obj,meth,[Args])`.
- Accessing an object field `Obj.field` when `Obj` is of class `p1.p2.c` becomes
  - `p1_p2_c:get_field(Obj)` or
  - `java:get(Obj,field)`.
- Similar functions available for accessing static members, and for setting the value of a field.

Where to find JavaErlang?

- Source code:
  `git://github.com/fredlund/JavaErlang.git`
- Compiled:
  `http://babel.ls.fi.upm.es/~fred/JavaErlang/`
- License is BSD

Demo time

Highlights

- 1-1 mapping between Erlang processes and Java threads (i.e, Erlang does not block because a Java call blocks)
- Flexible timeout handling – compare

  ```
  java:call_static(N,'java.lang.Thread',sleep,[10000]).
  ```

  with

  ```
  java:set_timeout(infinity).
  java:call_static(N,'java.lang.Thread',sleep,[10000]).
  ```

- Automatic boxing and unboxing of primitive method and constructor arguments

Limitations

- White-box testing: only **public** methods and attributes are accessible – easily fixable
- No automatic garbage collection for Java, i.e., Java does not know when no Erlang data structure refers to an object – should be possible to solve using the NIF library
- Speed. . .
- Implementing a Java class using Erlang

- Motivation:

Testing Java libraries using Erlang QuickCheck

- Motivation:
  - 80 student solutions written in Java

- Motivation:
  - 80 student solutions written in Java
  - Typical problems:
    Tree implementation, Finite state machine library, …

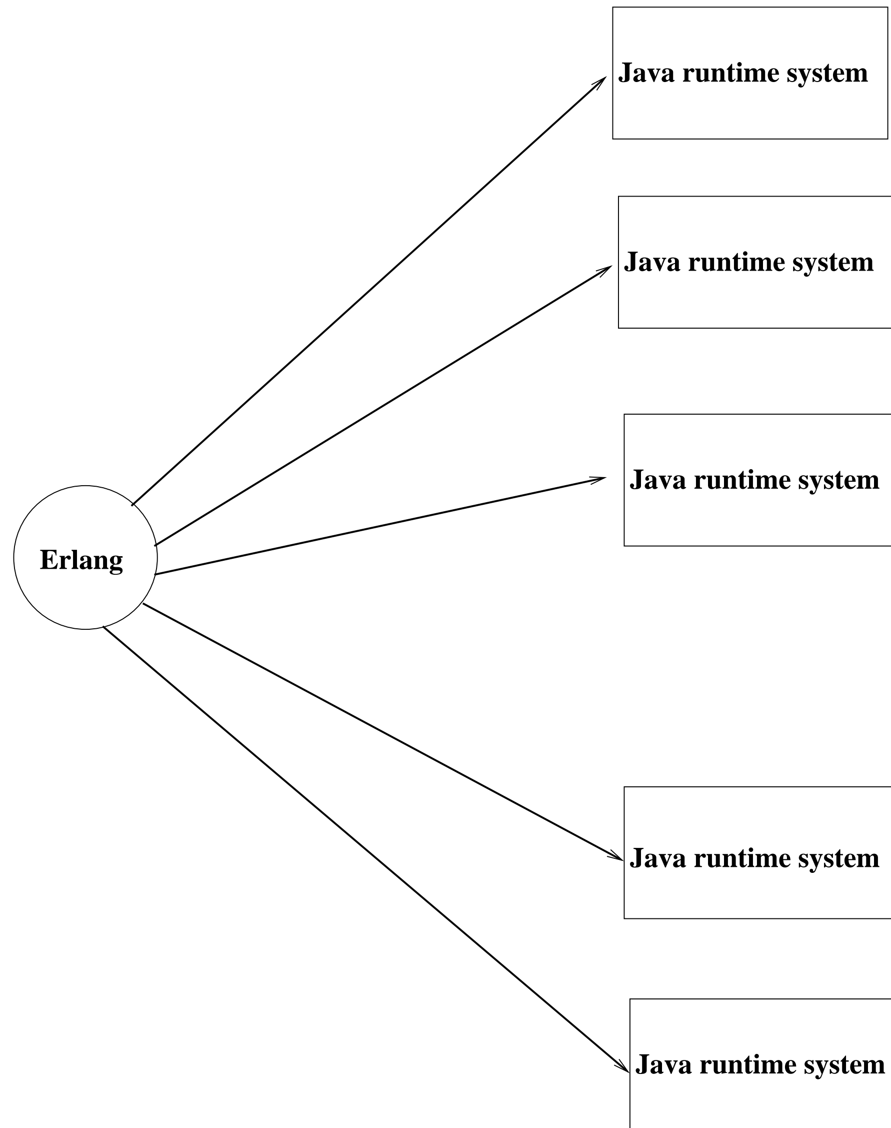
- Motivation:
  - 80 student solutions written in Java
  - Typical problems:
    Tree implementation, Finite state machine library, ...
  - We have to grade solutions, but have little time

- Motivation:
  - 80 student solutions written in Java
  - Typical problems:
    Tree implementation, Finite state machine library, . . .
  - We have to grade solutions, but have little time
  - And we have to **motivate** the grades :-) to the students

Testing Java libraries using Erlang QuickCheck

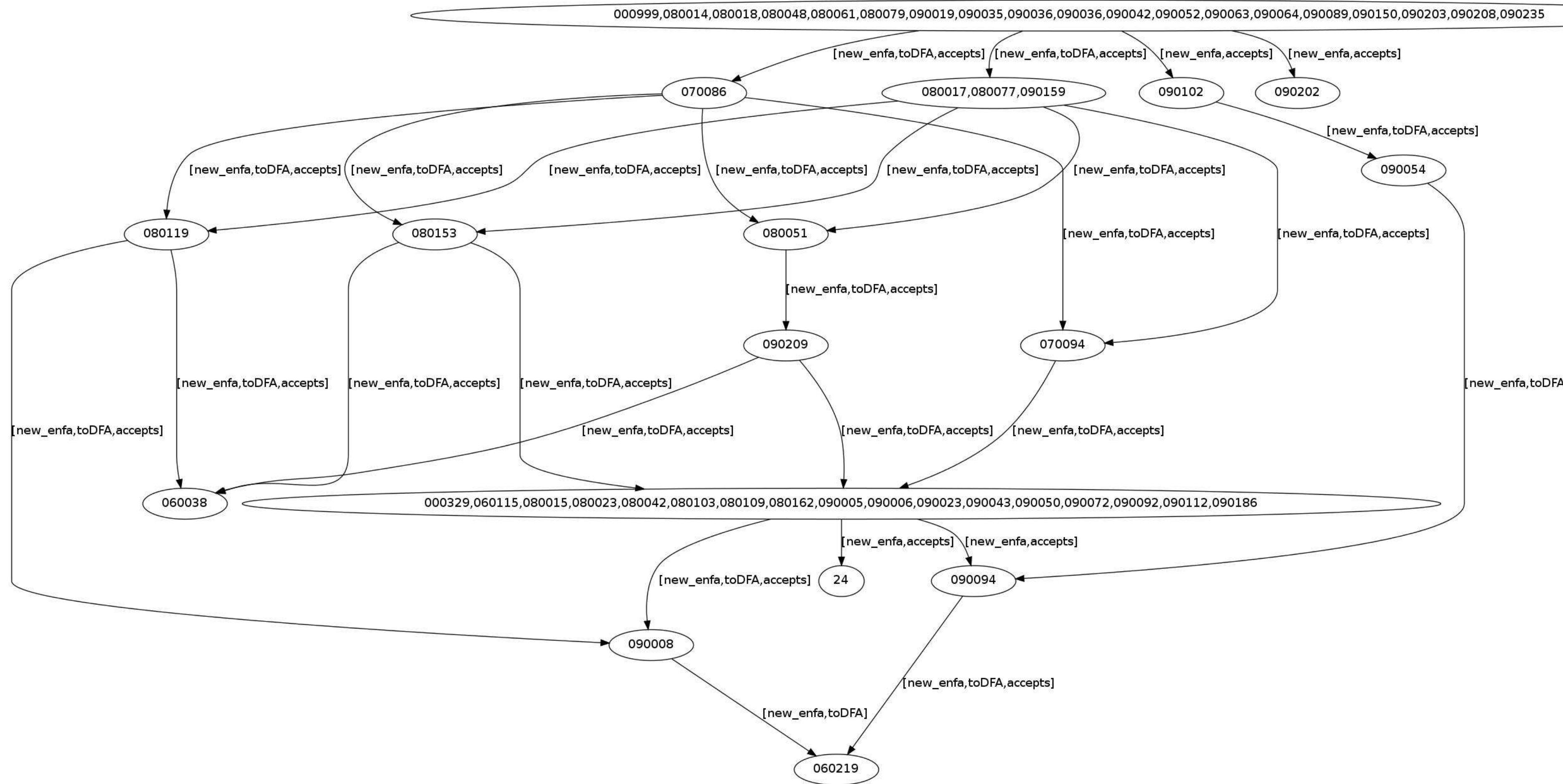
- Motivation:
  - 80 student solutions written in Java
  - Typical problems:
    Tree implementation, Finite state machine library, ...
  - We have to grade solutions, but have little time
  - And we have to **motivate** the grades :-) to the students
- Solution: use QuickCheck/Erlang to find errors, **rank** students, and **demonstrate** errors

Evaluation setup for ranking



Evaluation result 1: a ranking of the students

A grading of the students:



Evaluation results #2: demonstrate errors

A failing test sequence for a student:

```
Student 090232: next: no exception expected --
                exception 'MissingTransitionException' returned
```

```
Failing test:

{set,{var,2},{call,java,new,[{call,md,node_id,[]},'automata.DFA',[]]}}
    => {object,1,57678}
{set,{var,4},
     {call,java,new,[{call,md,node_id,[]},'automata.State',[false]]}}
    => void
{set,{var,8},{call,java,call,[{var,2},addState,[{var,4}]]}}
    => void
{set,{var,16},{call,md,addTransition,[{var,2},{var,4},a,{var,4}]}}
    => void
{set,{var,18},{call,java,call,[{var,2},setInitialState,[{var,4}]]}}
    => {java_exception,{object,10,57678}}
```

A practical example: testing a Java HashSet

`java.util.HashSet<E> interface:`

Method Summary

Methods

Modifier and Type	Method and Description
boolean	add(E e) Adds the specified element to this set if it is not already present.
void	clear() Removes all of the elements from this set.
Object	clone() Returns a shallow copy of this HashSet instance: the elements themselves a
boolean	contains(Object o) Returns true if this set contains the specified element.
boolean	isEmpty() Returns true if this set contains no elements.
Iterator<E>	iterator() Returns an iterator over the elements in this set.
boolean	remove(Object o) Removes the specified element from this set if it is present.
int	size() Returns the number of elements in this set (its cardinality).

Creating a QuickCheck model

- The library is *stateful* – we have to use a QuickCheck *state machine*
- The *state*:

  ```
  -record(state,{node_id=void,sets=[]}).
  ```

  (a Java node identifier, and a list of sets)

- A set in the model is a tuple

  ```
  {object_ref(), set()}
  ```

  where `object_ref()` is a Java reference to a set (in the concrete state) and `set()` is an Erlang set.

QuickCheck State Machine model

We have to provide the following functions:

- Generate tests:
  - `command` – generates a test
  - `precondition` – checks a test is "reasonable"
  - `next_state` – computes the next state for a test
- Interprete test outcome:
  - `next_state` – computes the next state in a test run
  - `postcondition` – checks that the return value of a call matches the expected value

Generating test

- A test is a sequence of commands
- One commands is generated by a call to the user-defined function `command(State::any())`
- A command is a call:
  `{call, Module::atom(), FunName::atom(), Args::[any()]}`
  representing a symbolic call `apply(Module,FunName,Args)`

Generating tests

```
command(State) ->
  eqc_gen:oneof
    (
    [{call,java,start_node,
        [[{java_exception_as_value,true}]]} ||
      State#state.node_id == void] ++
    %% Java exceptions returned as values

    [{call,java,new,
        [State#state.node_id,'java.util.HashSet',[]]} ||
      State#state.node_id =/= void] ++

    [{call,java,call,[Set,add,[nat()]]} ||
      {Set,_} <- State#state.sets] ++

    ...
    ).
```

Computing next states

Next states (during test generating **and** test execution) are computed by a call to the user-defined function

`next_state(State::any(),Var::any(),Call::call())`

- `State` is the state before the call was executed
- `Var` is the return value (test execution) **or** a symbolic variable (test generation)
- `call()` is a call {`Module,FunName,Args`}

Computing next states

```
next_state(State,Var,Call) ->
  case Call of
    {_,_,start_node,_} ->
      State#state{node_id = {call,?MODULE,node_id,[Var]}};

    {_,_,new,_} ->
      State#state{sets=[{Var,sets:new()}|State#state.sets]}

    {_,_,call,[Set,add,Elem]} ->
      {_,ESet} =
         lists:keyfind(Set, 1, State#state.sets),
      NewESet =
         sets:add_element(Elem,ESet),
      State#state
      {sets =
       lists:keyreplace(Set,1,State#state.sets,{Set,NewESet

    _ -> State
  end.
```

Checking return values of calls

Return values from executed test calls are checked against the model using a call to the user-defined function `postcondition(State::any(),Call::call(),Result::any())` where

- `State` is the state before the call was executed
- `Call` is the call
- `Result` is the result of the call

Checking return values of calls

```
postcondition(State,Call,Result) ->
  case Call of
    ...

    {_,_,_,[Set,contains,Elem]} ->
      {_,ESet} = lists:keyfind(Set,1,State#state.sets),
      Result == sets:is_element(Elem,ESet);

    _ ->
      not_exception(Result)
  end.

not_exception({java_exception,Exc}) -> false;
not_exception(_,_) -> true.
```

Demo time